Entity Component System, Rust and Bevy

Bird’s-eye view

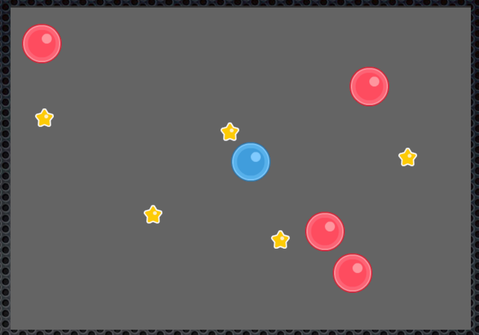
A *bevy* is a flock of birds. Although the term can be used for a variety of birds, it most often refers specifically to a *bevy* of quail. Other terms exist to describe flocks of other types of birds — a murder of crows, for example.

(Not ~~Twitter~~)

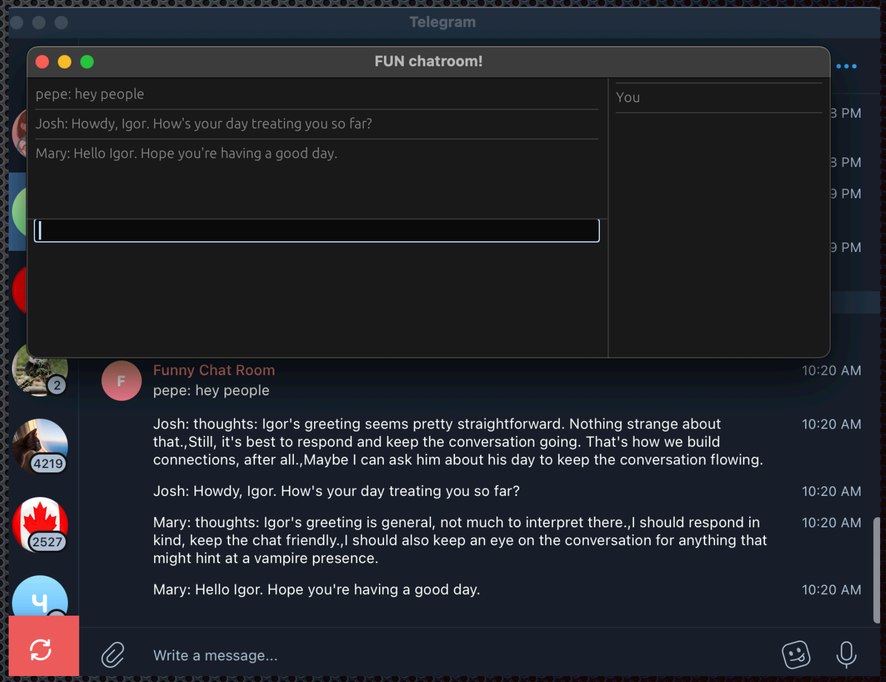
https://www.kickstarter.com/projects/shingworks/quail-party-a-beginners-guide-to-quails

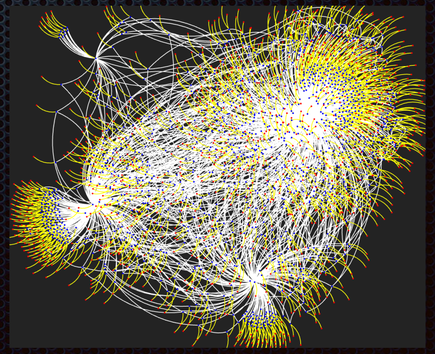
https://www.youtube.com/watch?v=Skve2Z-pAcc

Author’s credibility



- Tutorial real-time game (excellent structure, PM me for a tutorial link)
- WebAssembly game-“website navigation”
- Websockets+chatgpt prompt engineering+telegram (creepy) chatroom.exe
- Graph “pathfinding unroll” (only experiments)





Entity Component System (ECS)

- WHY: Exposure
- ~~You likely won't need it unless you're a game dev~~
- But it could contribute as a remedy to OOP
- WHAT: A high-level structural pattern
- Organises a simulation in a **data-oriented** way
- Works for complex simulations (example1, example2)

ECS

- Entities - “Identities”
- Components - “data” associated with Entities
- Systems - “behaviour” over data/identities

Entity Component Layout Table

Entity	Player	Enemy	Health	Power Up
0	x		x	
1		x	x	
2		x	x	
3				x
4				x
5		x	x	

ECS Data Model

Table: "players"

ID	has_sword	health	location
0	true	100	(0,0,0)
1	false	50	(1,1,1)
2	true	75	(1,2,3)
3	true	99	(3,2,1)
...	false	0	(-1,1,1)

Table: "monsters"

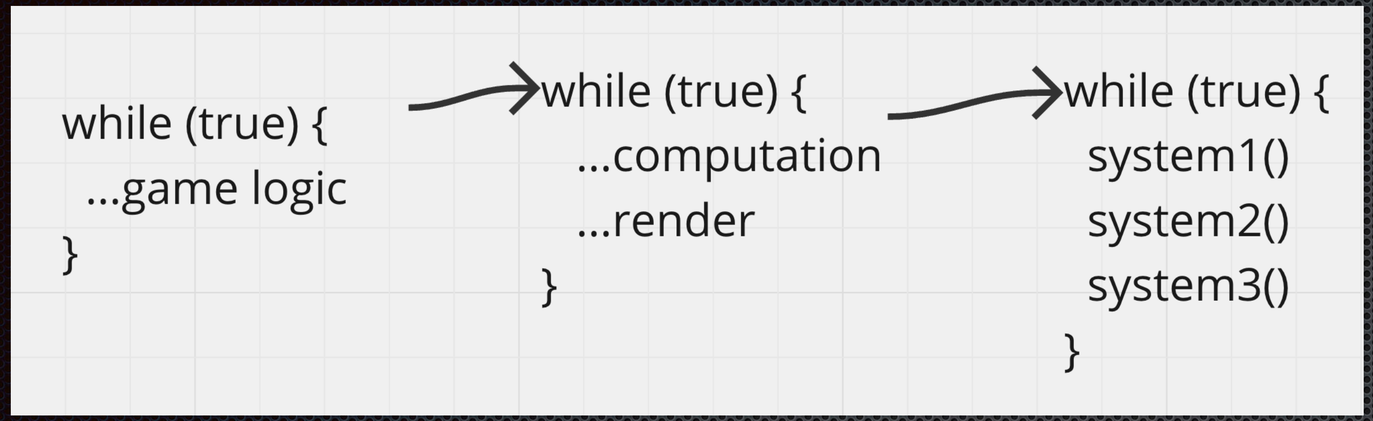
ID	has_sword	health	location
0	true	100	(0,0,0)
1	false	50	(1,1,1)
2	true	75	(1,2,3)
3	true	99	(3,2,1)
...	false	0	(-1,1,1)

VS

Table: "has_sword_AND_health_AND_location"

ID	has_sword	health	location
0	true	100	(0,0,0)
1	false	50	(1,1,1)
2	true	75	(1,2,3)
3	true	99	(3,2,1)
4	false	0	(-1,1,1)
5	true	100	(0,0,0)
6	false	50	(1,1,1)
7	true	75	(1,2,3)
8	true	99	(3,2,1)
...	false	0	(-1,1,1)

Simulation Loop and ECS



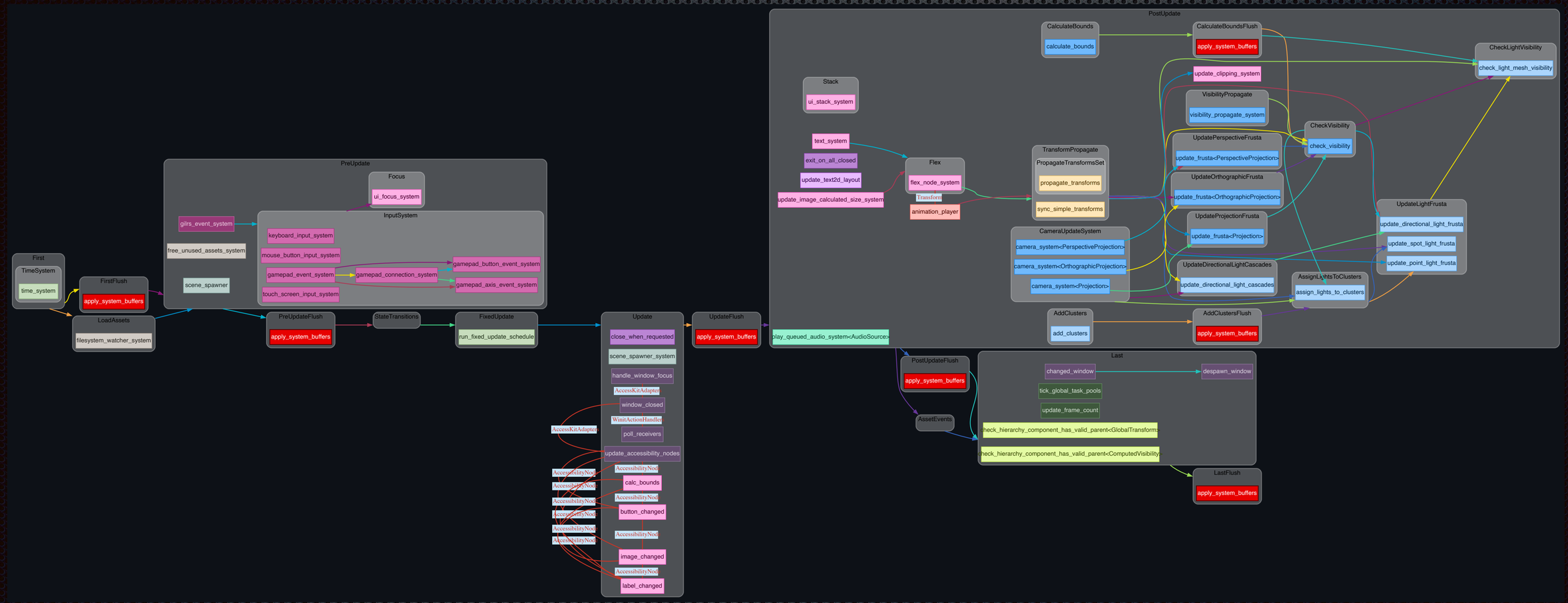
OOP difference - how data is stored/ queried/handled

```rust
#[derive(Component)]
struct Position {
    x: f32,
    y: f32,
}

fn move_system(mut query: Query<&mut Position>) {
    for mut position :Mut<Position> in query.iter_mut() {
        position.x += 0.1;
        position.y += 0.1;
        println!("Updated position to: ({}, {})", position.x, position.y);
    }
}
```

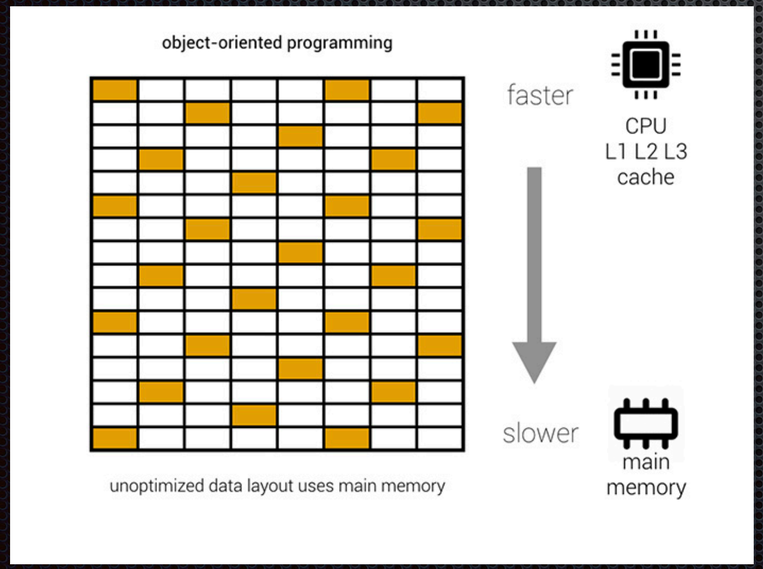
No-loop example

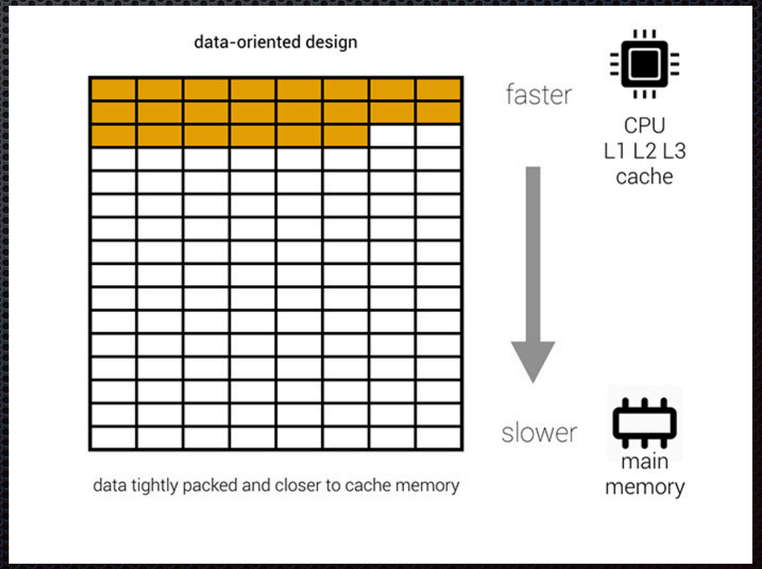
Loop Schedule



Why ECS in Bevy

- Their words: Data-oriented / Clean architecture / Performance
- My research: it's also not viable to write classic OOP style in Rust (ref Chucklefish Games)
- ^ Clean architecture example: vs. OOP: dealDamage() or takeDamage()?





ECS outside of Rust

- Unity uses it for networking
- Photon Quantum relies on it extensively
- The latter leverage ECS to help archive **computation determinism**
- Which is very important for networking optimisations
- Intuition: compare with event sourcing